RL-TR-94-28
Final Technical Report
April 1994

SOLUTIONS TO THE POLYINSTANTIATION PROBLEM

George Mason University

Sushil Jajodia and Ravi Sandhu

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-94-28 has been reviewed and is approved for publication.

APPROVED: Joseph V Giordano

JOSEPH V. GIORDANO
Project Engineer

FOR THE COMMANDER: John A Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control and Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3AB) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1994	Final - - - -

4. TITLE AND SUBTITLE
SOLUTIONS TO THE POLYINSTANTIATION PROBLEM

5. FUNDING NUMBERS
C - F30602-92-C-0002
PE - 33401G
PR - 1068
TA - 01
WU - P7

6. AUTHOR(S)
Sushil Jajodia and Ravi Sandhu

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
George Mason University
Center for Secure Info Systems & Dept of Info &
Software Systems Engineering
Fairfax VA 22030-4444

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3AB)
525 Brooks Road
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-94-28

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Joseph V. Giordano/C3AB/(315) 330-3681

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Polyinstantiation has generated a great deal of controversy lately. Some have argued that polyinstantiation and integrity are fundamentally incompatible, and have proposed alternatives to polyinstantiation. Others have argued about the correct definition of polyinstantiation and its operational semantics. The purpose of this report is to provide a tutorial on the subject. The report begins by reviewing the concept of polyinstantiation followed by a survey of various ways that have been proposed to deal with it. Finally, the impacts of various MLS DBMS architectures are discussed relative to polyinstantiation strategies. The report specifically discusses the differences between trusted subject and trusted computing base (TCB) subset with respect to polyinstantiation.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Database Security, Multilevel Security	56
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Acknowledgments

This work was partially supported by the U.S. Air Force, Rome Laboratory under the contract # F30602-92-C-0002. We are indebted to Joe Giordano for his support and encouragement which made this work possible.

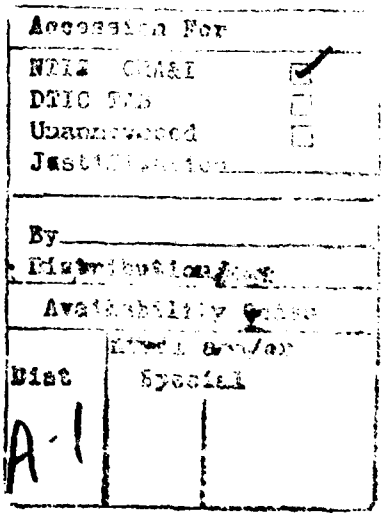

Table of Contents

Chapter 1

Introduction

What distinguishes a multilevel database from ordinary single level ones? In a multilevel world as we raise a user's clearance new facts emerge; conversely as we lower a user's clearance some facts get hidden. Therefore users with different clearances see different versions of reality. Moreover, these different versions must be kept coherent and consistent—both individually and relative to each other—without introducing any downward signaling channels.

The caveat of "no downward signaling channels" poses a major new problem in building multilevel secure database management systems (DBMSs) as compared to ordinary single-level DBMSs. Its considerations has lead to the notion of *polyinstantiation* in multilevel relations. The need for polyinstantiation was first identified in [17]; the term "polyinstantiation" was coined by the SeaView project [7]. Polyinstantiation comes in several different flavors [7, 16, 18, 19, 20, 22, 23, 25, 27, 28, 30, 31, 32, 33]. There are significant differences between these approaches and debate continues about the correct definition of polyinstantiation and its operational semantics. However, in each case polyinstantiation significantly complicates the semantics of multilevel relations (particularly for high users). As a result, recently some solutions have appeared which attempt to do away with polyinstantiation completely [3, 31, 38].

In this report, we first carefully review how the need for polyinstantiation arises in multilevel relations, followed by a survey of methods that have been developed for dealing with it.

The rest of this report is organized as follows. Chapter 2 reviews the concept of polyinstantiation from an intuitive point of view, with the objective of identifying the sources of polyinstantiation. Chapter 3 presents as a strawman a simple, but unacceptable solution. Chapter 4 introduces an example used to compare different approaches to polyinstantiation. Chapter 5 then discusses different approaches to

polyinstantiation. Chapter 6 discusses the architectural considerations that affect polyinstantiation. Chapter 7 concludes the report.

Chapter 2

What is Polyinstantiation?

In this chapter we show by means of examples how polyinstantiation arises. We assume that the readers are familiar with the basic concepts of the standard (single-level) as well as multilevel relations, as explained in [22].

In multilevel relations, access classes can be assigned to data stored in relations in four different ways. One can assign access classes to relations, to individual tuples in a relation, to individual attributes (columns) of a relation, or to individual data elements of the tuples of a relation. Polyinstantiation does not arise explicitly when access classes are assigned to relations or individual attributes of a relation and, therefore, we consider the cases when access classes are attached to tuples or the data elements themselves.

2.1 Types of Polyinstantiation

A multilevel relation is said to be *polyinstatiated* when it contains two or more tuples with the same apparant primary key values [7, 22]. There are two different types of polyinstantiation:

- Entity Polyinstantiation
- Attribute Polyinstantiation

Entity polyinstantiation occurs when a relation contains multiple tuples with the same apparant primary key values, but having different access class values for the apparant primary key. As an example, consider the relation SOD given in figure 2.1.

Starship		Objective		Destination	
Enterprise	U	Exploration	U	Talos	U
Enterprise	S	Spying	S	Rigel	S

Figure 2.1: A Multilevel Relation with Entity Polyinstantiation

In figure 2.1, as in most of our examples, each attribute in a tuple not only has a value but also a classification. We assume that the attribute Starship is the apparant primary key of SOD.

Now, the relation given above contains two tuples for the same starship Enterprise, resulting in entity polyinstantiation. These tuples can be regarded as pertaining to two different real-world entities or a single real-world entity. We cannot tell immediately by looking at the relation which is really the case.

The relation in figure 2.2 illustrates attribute polyinstantiation:

Starship		Objective		Destination	
Enterprise	U	Exploration	U	Talos	U
Enterprise	U	Spying	S	Rigel	S

Figure 2.2: A Multilevel Relation with Attribute Polyinstantiation

With attribute polyinstantiation, a relation contains two or more tuples with identical apparant primary key and the associated access class values, but having different values for one or more remaining attributes, as shown in the above relation. In the above multilevel relation, both tuples refer to a single Starship Enterprise; a S-user sees different values for its objective and destination.

As we indicated, explicit polyinstantiation can also occur with tuple level labeling instead of element level labeling. Let us consider the same example when access classes are associated with each tuple instead of each element. The S-user will see the mulltilevel relation shown in figure 2.3.

Notice that with tuple level labeling, we can no longer distinguish the entity polyinstantiation from attribute polyinstantiation. In our example relation, it is possible that both tuples relate to the same starship Enterprise; the U-tuple is merely

Starship	Objective	Destination	TC
Enterprise	Exploration	Talos	U
Enterprise	Spying	Rigel	S

Figure 2.3: A Multilevel Relation with Tuple-Level Labeling

the cover story. At the same time, it is also possible that there are two completely different starships; however, they have been given same name, possibly by error.

2.2 How polyinstantiation occurs

Either type of polyinstantiation can occur in basically two different ways which we call *visible* and *invisible* polyinstantiation respectively for mnemonic convenience.

1. Visible polyinstantiation occurs when a high user* attempts to insert data in a field which already contains low data. Since overwriting the low data in place will result in a downward signaling channel, the high data is inserted by creating a new tuple to store the high data.

2. Invisible polyinstantiation occurs in the opposite situation where a low user attempts to insert data in a field which already contains high data. Since rejecting the update is not a viable option because it establishes a downward signaling channel, the updated tuple is polyinstantiated to reflect the low update.

The next two sectionss make visible and invisible polyinstantiation clearer by considering some examples. The examples illustrate attribute polyinstantiation only; examples illustrating entity polyinstantiation can be constructed similarly.

2.3 Visible Polyinstantiation Example

Let us now consider a concrete example to make visible and invisible polyinstantiation clearer. Consider the following relation SOD where Starship is the apparent primary key.

*Strictly speaking we should be saying subject rather than user. For the most part we will loosely use these terms interchangeably. Where the distinction is important we will be appropriately precise.

Starship	Objective	Destination
Enterprise U	Exploration U	null U

Now consider the following scenario.

1. A U-user updates the destination of the Enterprise to be Talos. The relation is therefore modified as follows.

Starship	Objective	Destination
Enterprise U	Exploration U	Talos U

2. Next a S-user attempts to modify the destination of the Enterprise to be Rigel. Since we do not wish to deny entry of legitimate secret data, this update is not rejected. However, since we cannot overwrite the destination in place because that would create a downward signaling channel, we polyinstantiate and modify the relation to appear as follows, respectively for U- and S-users. Note that U-users see no change.

Starship	Objective	Destination
Enterprise U	Exploration U	Talos U

Starship	Objective	Destination
Enterprise U	Exploration U	Talos U
Enterprise U	Exploration U	Rigel S

What are we to make of this last relation given above. There are at least two reasonable interpretations.

- *Cover Story.* The destination of Talos may be a cover story for the real destination of Rigel. In this case the database is accurately mimicking the duplicity of the real world. There are, however, other ways of incorporating cover stories besides polyinstantiation. For example we may have two attributes, one for cover-story destination and one for the real destination. Debate on the relative merits and demerits of these techniques is outside the scope of this report.

- *Temporary Inconsistency.* We have a temporary inconsistency in the database which needs to be resolved. For instance the inconsistency may be resolved as follows: the S-user who inserted the Rigel destination latter logs in at the U level and nullifies the Talos value, so thereafter the relation appears respectively as follows to U- and S-users.

Starship	Objective	Destination
Enterprise U	Exploration U	null U

Starship	Objective	Destination
Enterprise U	Exploration U	Rigel S

It is important to understand that this scheme does not create a downward signaling channel from one subject to another. The nullification of the destination at the U level is being done by a U subject. One might argue that there is a downward signaling channel with a human in the loop. The human is however trusted not to let the channel be exercised without good cause. The real threat is to entity integrity: the U-user who executed step 1 of the scenario may again try to enter Talos as the destination, which brings us within the scope of the invisible polyinstantiation.

2.4 Invisible Polyinstantiation Example

Our example for invisible polyinstantiation is similar to the visible polyinstantiation example with the difference that the two update operations occur in the opposite order. So again consider the following relation SOD where Starship is the apparent primary key.

Starship	Objective	Destination
Enterprise U	Exploration U	null U

This time consider the following scenario.

1. A S-user modifies the destination of the Enterprise to be Rigel. The relation is modified to appear respectively as follows to U- and S users. Note that U-users see no change in the relation.

Starship	Objective	Destination
Enterprise U	Exploration U	null U

Starship	Objective	Destination
Enterprise U	Exploration U	Rigel S

2. A U-user updates the destination of the Enterprise to be Talos. We cannot reject this update on the grounds that a secret destination for the Enterprise already exists, because that amounts to establishing a downward signaling channel. Thus we have only one of the following two options left: We can overwrite the destination field in place at the cost of destroying secret data. This would give us the following relation for both U- and S-users.

Starship	Objective	Destination
Enterprise U	Exploration U	Talos U

For obvious reasons this alternative has not been seriously considered by most researchers. That leaves us the option of polyinstantiation which will modify the relation at the end of step 1 to the following for U- and S-users respectively.

Starship	Objective	Destination
Enterprise U	Exploration U	Talos U

Starship	Objective	Destination
Enterprise U	Exploration U	Talos U
Enterprise U	Exploration U	Rigel S

This is exactly the same relation as obtained at the end of step 2 in our visible polyinstantiation example. The possible interpretations are therefore similar, i.e., we either have a temporary inconsistency or a cover story. The temporary inconsistency can be corrected by having a U subject (possibly created by a S-user logged in at the U level) nullify the Talos destination. But the inconsistency may recur again and again.

Chapter 3

A Simple, but Unacceptable, Solution

There are two obvious "secure" alternatives to both visible and invisible polyinstantiations. These alternatives are secure in the sense of secrecy and information flow and preserve primary key requirements in multilevel relations; but unfortunately, they suffer from denial-of-service and other integrity problems.

1. Whenever a high user makes an update which violates the uniqueness requirement, we simply refuse that update.

2. Whenever a low user makes a change that conflicts with the uniqueness requirement, the conflicting high data is overwritten in place by the low data.

It is not difficult to see that this simple solution preserves the uniqueness requirement in multilevel relations. This solution is secure in the sense of secrecy and information flow. It is our view that while this solution may be acceptable in some specific situations, it is clearly unacceptable as a general solution; it can lead to serious denial-of-service and integrity problems. Therefore, we now look for other alternatives which do not suffer from these problems.

Chapter 4

An Example

Next chapter will describe several solutions to the polyinstantiation dilemma, some of which allow polyinstantiation in multilevel relations, while others seek to eliminate polyinstantiation completely. To appreciate the differences between various solutions, this chapter develops an example in more detail. Consider once again the relation SOD which has three attributes, Starship, Objective and Destination with Starship being the primary key.

Attribute	Classification Range
Starship	[U, U]
Objective	[U, S]
Destination	[U, S]
Tuple Class (TC)	[U, S]

Apparent Key: Starship

Figure 4.1: A scheme for the multilevel relation SOD.

If we were living in a single level world, for each starship there will be at most one tuple in this relation giving us that starship's unique objective and unique destination. For example, the tuple <Enterprise, Exploration, Talos> denotes that the starship Enterprise has set out to explore Talos. We say that this entire tuple gives us the mission of the Enterprise.

Next consider a multilevel relation which attempts to represent the same information, i.e., the objective and destination of starships, but in a multilevel world

where some facts are classified. Assume that there are just two levels, U for unclassified and S for secret. To further simplify the example let us say the Starship attribute is always unclassified. Therefore, the classification range of the Starship attribute has lower and upper bounds of U. On the other hand let the classification range of the Objective and Destination attributes have a lower bound of U and upper bound of S. Let us call the resulting schema SOD with the resulting schema summarized in figure 4.1. In this chapter, we will, for convenience, augment a relation scheme with a *tuple class* or TC attribute. This attribute is computed to be the least upper bound of the classifications of the individual data elements in the tuple. Thus, the value of TC gives the classification of the entire tuple.

The apparent primary key of SOD is specified as Starship. Intuitively this means that if only unclassified data is stored in SOD then Starship would be the actual primary key of the relation. Similarly, if only secret data is stored in the Objective and Destination attributes Starship would be the actual primary key. On the other hand if a mix of secret and unclassified data is stored in these attributes the actual primary key of SOD is Starship along with the attribute classifications. Instance 8 of figure 4.2 contains four tuples for the starship Enterprise. What makes each tuple distinct is the classification of the Objective and Destination attributes.

An instance of SOD is likely to contain different tuples at different levels. Therefore, it is important to distinguish between the U-instance of SOD, visible to unclassified users, and the S-instance, visible to secret users. As an user's clearance increases, it is reasonable to keep all previously visible information intact and perhaps add some new facts visible only at that level. To be concrete consider the U-instance of SOD given in figure 4.3. It contains exactly one tuple telling us that, as far as unclassified users are concerned, the starship Enterprise has set out to explore Talos. In figure 4.2 we enumerate eight different S-instances of SOD, all of which are consistent with the U-instance of figure 4.3. Their common property is that the single tuple of the U-instance appears in all eight S-instances. We regard each tuple in an instance of SOD as defining a mission for the starship in question. A U-instance of SOD allows only one mission per starship. S-instances on the other hand allow up to four missions per starship, three of which are secret and one unclassified.

We now demonstrate there is a practically useful and intuitively reasonable interpretation for each of the eight S-instances of figure 4.2. Consider each S-instance in turn as follows.

1. *The S-instance is identical to the U-instance.* There is therefore no secret aspect to the Enterprise. This is the simplest case and needs little explanation.

No.	Starship		Objective		Destination		TC
1	Enterprise	U	Exploration	U	Talos	U	U
2	Enterprise	U	Exploration	U	Talos	U	U
	Enterprise	U	Spying	S	Talos	U	S
3	Enterprise	U	Exploration	U	Talos	U	U
	Enterprise	U	Exploration	U	Rigel	S	S
4	Enterprise	U	Exploration	U	Talos	U	U
	Enterprise	U	Spying	S	Rigel	S	S
5	Enterprise	U	Exploration	U	Talos	U	U
	Enterprise	U	Exploration	U	Rigel	S	S
	Enterprise	U	Spying	S	Rigel	S	S
6	Enterprise	U	Exploration	U	Talos	U	U
	Enterprise	U	Spying	S	Talos	U	S
	Enterprise	U	Spying	S	Rigel	S	S
7	Enterprise	U	Exploration	U	Talos	U	U
	Enterprise	U	Spying	S	Talos	U	S
	Enterprise	U	Exploration	U	Rigel	S	S
8	Enterprise	U	Exploration	U	Talos	U	U
	Enterprise	U	Spying	S	Talos	U	S
	Enterprise	U	Exploration	U	Rigel	S	S
	Enterprise	U	Spying	S	Rigel	S	S

Figure 4.2: Eight S-instances of SOD.

Starship	Objective	Destination	TC
Enterprise U	Exploration U	Talos U	U

Figure 4.3: A U-instances of SOD.

In each of the next three cases there is a single tuple in the S-instance in addition to the tuple of the U-instance. This secret tuple defines a secret mission for the Enterprise in addition to its unclassified mission.

2. *The S-instance reveals the secret mission to be spying on Talos.* Presumably the unclassified exploration mission to Talos is a cover story to hide the secret spying mission. To maintain the integrity of the cover story, the Enterprise will probably expend resources on exploring Talos. Conceivably the bulk of its resources might be devoted to useful exploration of Talos with the secret spying mission added on as a low profile, low marginal cost and opportunistic effort. We obviously cannot resolve this issue without further knowledge about the real situation, such as a competent user might have. The main point is that the Enterprise does have two distinct missions: the unclassified one of exploring Talos and the secret one of spying there.

3. *The S-instance reveals the secret mission to be exploration of Rigel.* This case is very similar to the previous one in that only one attribute has a secret value. Clearly the desire to explore Rigel under cover of exploring Talos is a realistic one, not only in the national security arena but also in a competitive commercial context.

4. *The S-instance reveals the secret mission to be spying on Rigel.* This case is similar to the previous two in that there is only one secret mission. It is different in that the objective and destination of the secret mission are now both classified.

Each of the three preceding cases presents a distinctly different secret mission—secretly spying on Talos, secretly exploring Rigel and secretly spying on Rigel. These three secret missions do share the common property that exploring Talos is an acceptable unclassified cover story. The next three cases present situations where two of these three secret missions are concurrently in progress.

5. *The S-instance reveals two secret missions—to explore Rigel and to spy on Rigel.* Both secret missions are concerned with Rigel. Whether the principal one is

to explore it or spy there or the two missions are equally important, cannot be ascertained without further information. The secret exploration of Rigel may simply be a convenient damage control story, should the secret destination of the Enterprise be leaked. Conversely, spying on Rigel may be an opportunistic and relatively unimportant add on to its secret exploration.

6. *The S-instance reveals two secret missions—to spy on Talos and to spy on Rigel.* This is similar to the previous case and once again we cannot a priori decide which, if any, is the principal secret mission.

7. *The S-instance reveals two secret missions—to spy on Talos and to explore Rigel.* This may appear strange at first, but it is perfectly proper. For instance, there may be no life-forms on Rigel worth spying on while there are indications of vast quantities of Uranium. This S-instance does point out problems with simple rules such as "give the value with the highest classification for each attribute." Such a rule would manufacture the secret mission of spying on Rigel which does not exist in the relation.

As the reader may have guessed by now our final S-instance specifies that the three secret missions identified in instances 2, 3 and 4 are all concurrently in progress.

8. *The S-instance reveals three secret missions—to spy on Talos, to explore Rigel and to spy on Rigel.* As before, without further information and knowledge, we cannot say very much about the relation of these three secret missions to one another. All we know is that they share the same cover story of exploring Talos.

To summarize then eight S-instances of SOD can be partitioned into three classes as follows:

1. Instance 1 has no polyinstantiation and is therefore straightforward.

2. Instances 2, 3, and 4 are also relatively straightforward. Instance 2 has a cover story for the objective, but the U destination is correct. Instance 3 on the other hand has a cover story for the destination, while the objective is correct. Instance 4 has a cover story for both the destination and the objective.

3. Instances 5, 6, 7, and 8 are confusing to interpret if it is assumed that the higher level data correctly represent the real world. Nonetheless, it is possible to give a meaningful and consistent interpretation and update semantics for both the objective and the destination.

Chapter 5

Solutions to the Polyinstantiation Problem

There are a number of different approaches to implementing polyinstantiation in a database management system, reflecting divergent perspectives on the meaning and uses of polyinstantiation within an MLS environment. Each perspective has its proponents and detractors, and each is suited to particular types of applications. It is not our intent to promote certain approaches or to dismiss others, but instead to discuss the perspective motivating each of them. It is our belief that different organizations and real-world enterprises will choose to model their understandings of multilevel data in distinct ways. Our goal here is to present multiple approaches and their rationales so that each organization may choose the most appropriate implementation for its requirements.

This chapter starts with those approaches that view polyinstantiation (and the concomitant addition of tuples) as an integral part of an MLS database. Next, the chapter presents strategies that compose new tuples to answer queries based on the security levels of underlying tuples. Finally, it discusses approaches that include explicit restrictions on users' views of data.

5.1 Propagation of Polyinstantiated Tuples

One perspective in dealing with the tension between multilevel security and data semantics is to regard polyinstantiation as an inevitable and integral part of multilevel secure information. Users at different security levels may see different attribute values for the same real-world tuple (e.g., Secret vs. Unclassified objectives for the same starship), and the users must be allowed to update these values differently. This perspective leads to an approach to polyinstantiation in which new tuples are added to reflect the combinatorial explosion of attribute values. For simplicity, we will call

this approach the *propagation approach* to polyinstantiation.

The propagation approach faces two key challenges: (1) ensuring that keys still function to identify distinct real-world entities, and (2) controlling the propagation of tuples to include only meaningful combinations of attribute values. The first challenge is met by augmenting the apparent key with a security level and enforcing the standard key uniqueness property over this augmented key. The second challenge is more complex and researchers are still debating which types of combinations are meaningful. In general, multivalued dependencies (see [5] for more detailed explanation) are used to define the particular combinations allowed by a specific solution. While many variants are possible, the SeaView project [7, 8, 9, 26, 27, 28] and the proposed modifications of Jajodia and Sandhu [18] provide the basis of this approach. First we present the original SeaView approach, then Jajodia's and Sandhu's proposed modification, and finally some new techniques proposed by the SeaView project.

The SeaView project began as a joint effort by SRI International and Gemini Computers with the goal of designing and prototyping a MLS relational database management system that satisfies the Trusted Computer System Evaluation Criteria for Class A1 [1]. Currently the project is in the final phase of a prototype implementation using GEMSOS as the underlying trusted computing base along with the ORACLE relational DBMS [27].

SeaView solves the problem of polyinstantiation of key attributes themselves by defining an entity integrity property. This property requires all attributes in a key to be uniformly classified. That is, for any instance R_c of a multilevel relation schema, for any tuple $t \in R_c$, and for any attributes A_i and A_j in the apparent primary key K_R of R, $t[C_i] = t[C_j]$. Notice that this means that it is possible simply to define a single attribute C_K to represent the classification level of all attributes in the apparent primary key. Further, no tuples may have null values for key attributes. This restriction ensures that keys can be meaningfully specified and checked for uniqueness. In addition, all non-key classification attributes must dominate C_K. This restriction guarantees that if a user can see any part of a tuple, then he or she can see the key.

To meet the first challenge, that of using keys to determine when tuples model distinct real- world entities, SeaView defines a polyinstantiation integrity property. The formulation of polyinstantiation integrity in SeaView consists of two distinct parts. The first part consists of a functional dependency component whose effect is to prohibit polyinstantiation within the same access class. The second part consists of a multivalued dependency requirement.

SeaView Polyinstantiation Integrity Property: A multilevel relation R_c satisfies ***polyinstantiation integrity*** (PI) if and only if for every R_c there are for all $A_i \in K_R$

1. $K_R, C_K, C_i \rightarrow A_i$

2. $K_R, C_K \rightarrow\!\!\rightarrow A_i, C_i$

The PI property can be regarded as implicitly defining what is meant by the primary key in a multilevel relation. The primary key of a multilevel relation is $K_R \cup C_K \cup C_R$ (where C_R is the set of classification attributes for data attributes not in K_R) since from PI it follows that the functional dependency $K_R \rightarrow A_R$ holds (where A_R consists of all attributes that are not in K_R).

Of the eight instances defined in figure 4.2, this definition of polyinstantiation integrity allows *only two* combinations of these eight instances within a single relation scheme [18]. Specifically a SeaView relation can accommodate either instances 1, 2, 3, and 8 or instances 1 and 4 within a single scheme in the absence of the uniform classification constraint. SeaView admits only instances 1 and 4 if the Objective and Destination attributes are uniformly classified (i.e., either both are classified U or both S).

The inclusion of the multivalued dependency in the definition of polyinstantiation integrity means that one update may result in a number of tuples being added to the relation. To illustrate, consider the situation in which an S user attempts to go from S instance 1 to S instance 4 in figure 4.2 by inserting the secret tuple specifying the secret mission of spying on Rigel. SeaView will interpret this as a request to go from S instance 1 to S instance 8, thereby manufacturing two additional missions for the Enterprise. Unfortunately, this increases the potential for such additional information that may not reflect true data to be retrieved from the database by users with higher clearances. It is easy to see that, in the worst case, the number of manufactured tuples materialized grows at the rate of $|\text{security-lattice}|^k$ where k is the number of non-key attributes in the relation. For example, figure 5.1 shows a TS-instance of a relation similar to SOD, except that it has a range of four security levels for the Objective and Destination attributes. The particular TS instance shown there describes four missions for the Enterprise, one each at the unclassified, confidential, secret and top-secret levels. The definition of polyinstantiation integrity in SeaView requires that this information be represented by the sixteen missions shown in figure 5.2. Users with clearance U, C, S, and TS will respectively see 1, 4, 9, and 16 missions with the SeaView approach.

In [18], Jajodia and Sandhu proposed dropping the multivalued dependency from the polyinstantiation integrity property defined in the SeaView model. They argued that the multivalued dependency prohibits the existence of relation instances that are desirable in practice. Specifically, it is possible to accommodate all eight

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Enterprise	U	Mining	C	Sirius	C	C
Enterprise	U	Spying	S	Rigel	S	S
Enterprise	U	Coup	TS	Orion	TS	TS

Figure 5.1: A TS-instance of SOD with 4 missions.

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	Talos	U	U
Enterprise	U	Exploration	U	Sirius	C	C
Enterprise	U	Mining	C	Talos	U	C
Enterprise	U	Mining	C	Sirius	C	C
Enterprise	U	Exploration	U	Rigel	S	S
Enterprise	U	Mining	C	Rigel	S	S
Enterprise	U	Spying	S	Talos	U	S
Enterprise	U	Spying	S	Sirius	C	S
Enterprise	U	Spying	S	Rigel	S	S
Enterprise	U	Exploration	U	Orion	TS	TS
Enterprise	U	Mining	C	Orion	TS	TS
Enterprise	U	Spying	S	Orion	TS	TS
Enterprise	U	Coup	TS	Talos	U	TS
Enterprise	U	Coup	TS	Sirius	C	TS
Enterprise	U	Coup	TS	Rigel	S	TS
Enterprise	U	Coup	TS	Orion	TS	TS

Figure 5.2: The SeaView materialization with 16 missions.

instances of figure 4.2. Jajodia and Sandhu give formal operational semantics for update operations in multilevel relations in [22, 23].

Based on this proposal, the SeaView team began a reexamination of the SeaView definition of polyinstantiation integrity. In [28], Lunt and Hsieh develop a semantics for the basic database manipulation operations (insert, update, and delete). Based on these semantics, they propose a different definition for polyinstantiation integrity consisting of two separate pieces: a state property containing the same functional dependency component and a transition property concerning a new dynamic multivalued dependency component. Although Lunt and Hsieh do not define the latter property precisely, the basic idea can be illustrated informally by way of an example from [28].

Consider the multilevel relation scheme $R(A_1, C_1, A_2, C_2, A_3, C_3, TC)$ where each A_i is an attribute, each C_i is the classification attribute for A_i, and TC is the tuple class attribute. The attribute A_1 is the apparent primary key of R. An instance R_c at a classification level c is assumed to satisfy the two constraints of the PI property.

Now, consider the following relation instance R_U:

A_1	C_1	A_2	C_2	A_3	C_3	TC
a	U	b	U	x	U	U

Suppose a Confidential user changes the value of A_2 to d, as shown here:

A_1	C_1	A_2	C_2	A_3	C_3	TC
a	U	b	U	x	U	U
a	U	d	C	x	U	C

Under the update semantics of [28], whenever an update involves some, but not all, of the nonkey attributes, certain dynamic multivalued dependencies are enforced in the multilevel relations. In the example, the dynamic multivalued dependencies are:

$$A_1, C_1 \twoheadrightarrow A_2, C_2 | A_3, C_3$$

where the notation $X \twoheadrightarrow Y|Z$ denotes the multivalued dependencies $X \twoheadrightarrow Y$ and $X \twoheadrightarrow Z$.

Next, suppose a Top Secret user updates the value of A_3 to equal "v." As before, since this update involves some (but not all) of the nonkey attributes, the dynamic multivalued dependency property causes two more tuples to be added to the relation:

A_1	C_1	A_2	C_2	A_3	C_3	TC
a	U	b	U	x	U	U
a	U	d	C	x	U	C
a	U	b	U	v	TS	TS
a	U	d	C	v	U	TS

At this point suppose a Secret user changes the value of the second attribute to "q." The following relation instance results:

A_1	C_1	A_2	C_2	A_3	C_3	TC
a	U	b	U	x	U	U
a	U	d	C	x	U	C
a	U	b	U	v	TS	TS
a	U	d	C	v	U	TS
a	U	q	S	x	U	TS
a	U	d	C	v	U	TS

As stated in [28], the way in which an update occurs determines whether or not the multivalued dependency should be enforced. Essentially, if two or more attributes were updated in a single update statement, the multivalued dependency would not be enforced. However, if the two attributes were updated in two independent operations, the multivalued dependency would be enforced.

This dynamic approach is not yet formalized, nor is it being incorporated in the SeaView prototype.

5.2 Derived Values

A second perspective on polyinstantiation is that although a multilevel relation may have several tuples for the same real-world entity, there should be only one such

tuple per classification level. Instead of a classification level C_i for each attribute A_i, the schema R_c includes a single classification level for each tuple, TC. When a user wants to update only certain attributes at a particular level, the values of the other attributes are derived from values at lower security levels.

Consider the following relation SOD where Starship is the key:

Starship	Objective	Destination	TC
Enterprise	Exploration	Talos	U

Now suppose an S-user wishes to modify the destination of the Enterprise to be Rigel. He or she can simply do so by inserting a new Secret tuple to SOD as follows:

(Enterprise, $\hat{U}$, Rigel, S)

The symbol $\hat{U}$ is to be interpreted as follows: For this S tuple the value of the Objective field is identical to the corresponding value in the U tuple of SOD. As a consequence, when a S user asks for the SOD relation to be materialized, he or she will see the following:

Starship	Objective	Destination	TC
Enterprise	Exploration	Talos	U
Enterprise	Exploration	Rigel	S

The relation will appear unchanged to the U-user.

The Lock Data Views (LDV) project [16] follows this *derived data approach.*

The derived data approach has been implemented for the U.S. Transportation Command Air Mobility Command MLS Global Decision Support System (GDSS) [29]. This implementation, the MLS GDSS , limits polyinstantiation in a multilevel relation to at most one tuple per security class. Information is labeled at one of two levels, U or S. The design is based on the organization's assumption that when S and U data are integrated into a single S response the S data takes precedence over the U data. This design can be extended to environments with more than two strictly ordered security levels. Organizations for which this strict hierarchical rule does not apply, such as many compartmented environments, would have to incorporate substantial changes into this design in order to use it.

In the MLS GDSS application, trusted application software functionally extends the commercial off-the-shelf (COTS) MLS DBMS to manage tuple level polyinstantiation. Before inserting an S tuple, the trusted software ensures that a U-tuple exists with the same key. If it does not exist, the insertion of an S tuple is not permitted. If a U-tuple with the same apparent primary key does exist, the trusted application software examines each S-tuple attribute value, except the apparent key value, and determines if it replicates the attribute's value in the U-tuple. If so, the value is not replicated in the S-tuple but instead set to null, minimizing data replication. The U-tuple thus serves as the foundation upon which the S-tuple is built. The MLS GDSS solution is best explained with several examples. Consider the following relation:

Starship	Objective	Destination	TC
Enterprise	Exploration	Talos	U

Now suppose an S-user wishes to modify the destination of the Enterprise to Rigel. The S- user directs the system, through the trusted software, to insert an S-tuple into the SOD as follows:

S-USER:

Insert into

(Starship, Objective, Destination)

Values ('Enterprise', 'Exploration', 'Rigel')

The U and S tuples are now stored in the relation as:

Starship	Objective	Destination	TC
Enterprise	Exploration	Talos	U
Enterprise	Null	Rigel	S

By reducing the replication of data across polyinstantiated tuples, the probability of maintaining the integrity of the database improves. Additionally, except for the key value, the sensitivity level of all attribute values contained within the stored tuple are equivalent to the TC value. Given this equivalence to the TC value, trusted application software derives attribute value labels from the TC value. Users operating at the U-level are presented with a display showing the derived attribute value labels as follows:

Starship	Objective	Destination
Enterprise U	Exploration U	Talos U

Users operating at the S level are presented with a single composite display of a materialized tuple. This materialized tuple comprises S and U data as follows:

Starship	Objective	Destination
Enterprise U	Exploration U	Rigel S

One of the major impacts of the polyinstantiation approach as implemented in the MLS GDSS, involves the DBMS join operator at the S level. Figure 5.3 illustrates the simplest form of the problem. A typical join operation between two tables matches and retrieves rows based on the primary key Starship. In order to retrieve data residing at the same security level, and thus permit proper collapsing of the rows into a materialized tuple, the join is further qualified by the row's security label attribute TC.

S-USER:

```
Select *
FROM Table1, Table2
where Table1.Starship = Table2.Starship
and Table1.TC = Table2.TC
```

An important functional requirement in the MLS GDSS is that S users expect to see S data as the end product of a retrieval, if S data exists; otherwise, U data is returned. Case 1 in figure 5.3 shows a join between two tables that produces the correct materialized tuple for an S user. Case 2 illustrates the anomaly associated with the join. In this case, table 2 contains only U data. Since the query requires that the tuple labels match, the query does not return the S row of table 1 joined with the U row of table 2. Thus, if data does not exist at the same security levels in each table, then S information may be lost during the join operation.

In this simplified example, one might argue that removing the qualification that the tables be joined by tuple labels would permit joins. Doing this would return two rows in Case 2, one containing only U information, the other containing S and U information. If this approach were taken, the tuple materialization process becomes more complex and would need to extract multiple tuple labels and assign them to

Case 1:

Starship	Objective	Destination	TC
Enterprise	Exploration	Talos	U
Enterprise	null	Rigel	S

Starship	Type	Propulsion	TC
Enterprise	Starship	Photon	U
Enterprise	Battlestar	Queller Drive	S

Starship		Objective		Destination		Type		Propulsion	
Enterprise	U	Exploration	U	Rigel	S	Battlestar	S	Queller Drive	S

Case 2:

Starship	Objective	Destination	TC
Enterprise	Exploration	Talos	U
Enterprise	null	Rigel	S

Starship	Type	Propulsion	TC
Enterprise	Starship	Photon	U

Starship		Objective		Destination		Type		Propulsion	
Enterprise	U	Exploration	U	Talos	U	Starship	U	Photon	U

Figure 5.3: Joins in GDSS

the appropriate columns in the row that was returned. Also, the join example shown in Case 1 would result in four rows of data returned from the server, instead of just two. The complexity of the problem and the work required of the DBMS server would increase significantly as more tables are joined. Database server performance would decrease accordingly, perhaps to unacceptable levels.

In order to ensure the correct materialization of a logical joined tuple, the MLS GDSS system does not currently use the join capabilities of the COTS MLS DBMS. Instead, tuples are selected from individual tables, and then joined outside the DBMS by trusted application software. While this operation does result in some processing overhead, it ensures that data are not accidentally excluded from the S-user.

5.3 Visible Restrictions

The third perspective on polyinstantiation is that users are aware that data are restricted to certain levels. In practice, this perspective means that users know the levels of data that they can see and update. The goal is to provide a more "honest" database without compromising security. This perspective can lead to many different strategies; this chapter presents four different approaches.

5.3.1 The Belief Approach

One approach to polyinstantiation is motivated by the idea that data at each level reflect the "beliefs" of users at that level about the real world [35]. For simplicity, we will call this work the *belief approach*. The belief approach differentiates between data that a user sees and data that a user believes. Updates reflect beliefs about the real world; they are regulated by the following property.

Update Access Property: Data at a particular level can only be inserted, modified, or deleted by users at that level.

Thus, data at each level reflects the beliefs of the users who maintain it. Users may see the data that they believe as well as data believed by users at lower levels (i.e., users see all data that they could read under the Bell-LaPadula model).

At the heart of this property is a model that takes a stand between entity- and attribute-level polyinstantiation. Keys may be classified at a different level than other attributes within the same tuple, but all non-key attributes within a single tuple share a classification level.

Starship	K_c	Objective	Destination	T_c
Voyager	U	Shipping	Mars	U
Enterprise	U	Exploration	Vulcan	U
Enterprise	U	Diplomacy	Romulus	C
Zardor	S	Warfare	Romulus	S
Voyager	S	Spying	Rigel	S

Figure 5.4: Example of SOD in the Belief Model

Given a relation schema R, the multilevel relation R_c used in the belief model includes two additional classification attributes: a key classification level (K_c) and a tuple classification level (T_c). The model imposes two restrictions:

1. In any tuple, T_c must dominate K_c.

2. For the set of key attributes K and for all non-key attributes $A_i, \ldots, A_n$ in R_c,

$$K, K_c, T_c \rightarrow A_i, \ldots, An$$

Intuitively, then, tuples with the same values for key attributes but different key classification levels refer to different real-world entities. Tuples that are identical in key attributes and key classification levels but differ in tuple classification levels represent different beliefs about the same real-world entities. To maintain this distinction, users at a particular level are not allowed to reuse key attribute values for new entities.

Given the relation SOD in figure 5.4, U-users believe the first and second tuples. C-users believe the third tuple, and S-users believe the fourth and fifth tuples. Since the first and fifth tuples refer to the same real-world entity, the first tuple indicates a cover story (or simply erroneous information on the part of a U-user).

The second and third tuples in figure 5.4 refer to the same real-world starship, but U- and C- users have different beliefs about its objective and destination. The first and fifth tuples refer to different starships.

U-users can see only the first two tuples in figure 5.4, C-users can see the first three tuples, and S-users can see all five tuples.

Although users are allowed to see all tuples at levels dominated by their belief levels, the query language includes the optional keyword BELIEVED BY to allow users to further restrict queries. Thus, S-users can ask to see all allowable tuples, or only those believed by C- and S-users, etc.

So, the query "Display the destination of all starships named Enterprise" is expressed as

```
SELECT        Destination
FROM          SOD
WHERE         Starship = "Enterprise"
BELIEVED BY   ANYONE
```

The result of this query when issued against the relation in figure 5.4 is:

Destination	TC
Vulcan	U

for a U-user, and

Destination	TC
Vulcan	U
Romulus	C

for all users at levels C or higher.

The query "Display the beliefs of U-users as to the destination of all starships named Enterprise" is expressed as:

```
SELECT        Destination
FROM          SOD
WHERE         Starship = "Enterprise"
BELIEVED BY   U
```

The result of this query when issued against the relation in figure 5.4 is:

Destination	TC
Vulcan	U

for all users.

The query "Display the classification level and destination of all starships named Voyager" is expressed as:

SELECT K_c, Destination
FROM SOD
WHERE Starship = 'Voyager'
BELIEVED BY ANYONE

The result of this query when issued against the relation in figure 5.4 is

K_c	Destination	T_c
U	Mars	U

for U- and C-users, and

K_c	Destination	T_c
U	Mars	U
S	Rigel	S

for all users at levels S or higher.

5.3.2 The Insert-Low Approach

Another variation of explicit restriction, the *insert-low approach*, has been adopted by the SWORD project at the Royal Signals and Radar Establishment in England [38]. Briefly, this approach works as follows:

Each relation is assigned at the time of its creation a *table usage classification*, abbreviated as table class . Each attribute is assigned a *column classification* which must dominate the table class.

The purpose of the table class is two-fold: First, any insertion or deletion of tuples in a relation can be made by those users whose clearances equal the table class of the relation. Second, the table class controls exactly how the updates involving an access class that dominates the table class are made to the relation. This will be explained in greater detail below.

Consider once again the relation schema SOD. Say the table classification of SOD is U.

A typical instance of SOD is given as follows:

Starship		Objective		Destination	
Enterprise	U	Exploration	U	Talos	U
Voyager	U	Spying	S	Rigel	TS

In this case, SWORD will show the entire relation to TS users, while for those at lower levels SWORD will substitute <not cleared> whenever a user has insufficient clearance to view a value. Thus, for example, a C-user will see the following instance:

Starship		Objective		Destination	
Enterprise	U	Exploration	U	Talos	U
Voyager	U	<not cleared>		<not cleared>	TS

To see how SWORD avoids tuple polyinstantiation, consider once again the relation SOD with U as its table class. Suppose the initial database state is as follows:

Starship		Objective		Destination	
Enterprise	U	Exploration	U	Talos	U

Suppose some U-user inserts the tuple (Voyager, S, Spying, U, Talos, U) to SOD. SWORD allows lower-level users to insert values at higher levels as long as the attribute value classifications are dominated by the appropriate column classification. In this example, the column classification for Starship would have to be S or higher. Furthermore, since the table classification of SOD is U, this constitutes a legal insertion, and as a result U-users and S- users will see the following states respectively:

Starship		Objective		Destination	
Enterprise	U	Exploration	U	Talos	U
<not cleared>		Spying	U	Talos	U

Starship		Objective		Destination	
Enterprise	U	Exploration	U	Talos	U
Voyager	S	Spying	U	Talos	U

At this point, suppose a U-user wants to make an insertion (Freedom, U, Mining, U, Mars, U) to SOD. Since the Starship attribute of all tuples in SOD are not visible to the U-user, there is always a possibility that the Starship value of the tuple to be inserted equals that of the existing high tuple, leading to attribute polyinstantiation (or tuple polyinstantiation in the case of attributes constituting the primary key). SWORD avoids this by prohibiting U-users from inserting or modifying values in this attribute. In the case of key attributes, like Starship, this means that all further insertions by U-users are forbidden. However, since the table classification is U, only U-users can insert tuples into SOD. As a consequence, ***no further insertions can be made to SOD at all.*** In SWORD applications, then, the column classifications for all attributes constituting the primary key must equal the table class, or users may be able to prohibit future insertions.

The following instance illustrates in more detail how attribute polyinstantiation is avoided in SWORD:

Starship		Objective		Destination	
Enterprise	U	Exploration	U	Talos	U

Next suppose a TS-user wishes to modify the destination of the Enterprise to be Rigel. This is accomplished in two steps. First, the TS-user must log in as a U-user and change the classification of Talos from U to TS. Having done so, the TS-user can log in at his level and then make the desired update. As a result, the U instance and TS instance will become as follows:

Starship		Objective		Destination	
Enterprise	U	Exploration	U	<not cleared>	TS

Starship	Objective	Destination
Enterprise U	Exploration U	Rigel TS

Given the database state shown immediately above, suppose an S-user wants to insert a Secret destination for Enterprise. He may do so by first logging in as a U-user, changing the classification of the attribute Destination from TS to S. As a result of this change, all users, *including* the TS-user, will see the following relation:

Starship	Objective	Destination
Enterprise U	Exploration U	<not cleared> S

Now, the S-user can log in at classification level S and make the appropriate change.

5.3.3 Prevention

The third variation of explicit restriction relies on preventing polyinstantiation completely. In [24, 31, 32], Jajodia and Sandhu describe three basic techniques for eliminating entity polyinstantiation.

1. ***Make all the keys visible.*** In this method the apparent primary key is required to be labeled at the lowest level at which a relation is visible. For example, suppose that the designer requires that all keys be unclassified. Consequently, the following relation:

Starship	Objective	Destination	TC
Enterprise U	Exploration U	Talos U	U
Enterprise S	Spying S	Rigel S	S

would be forbidden. Note that the two relations, called USOD and SSOD, in figures 5.5 and 5.6, represent the same information.

In other words, USOD and SSOD horizontally partition the original SOD relation, with all the U-Starships in USOD and all the S-Starships in SSOD.

UStarship	Objective	Destination	TC
Enterprise U	Exploration U	Talos U	U

Figure 5.5: USOD

SStarship	Objective	Destination	TC
Enterprise S	Spying S	Rigel S	S

Figure 5.6: SSOD

2. ***Partition the domain of the primary key.*** Another way to eliminate entity polyinstantiation is to partition the domain of the primary key among the various access classes possible for the primary key. For our example, suppose that the application requires that starships whose names begin with A-E are unclassified, starships whose names begin with F-T are secret, and so on. Whenever a new tuple is inserted, the system enforces this requirement as an integrity constraint. In this case the secret Enterprise must be renamed, perhaps as follows.

Starship	Objective	Destination	TC
Enterprise U	Exploration U	Talos U	U
Enterprise S	Spying S	Rigel S	S

   The DBMS can now reject any attempt by a U-user to insert a Starship whose name begins with F-Z, without causing any information leakage or integrity violation.

3. ***Limit insertions to be done by trusted subjects.*** A third way to eliminate entity polyinstantiation is to require that all insertions are done by a system-high user, with a write-down occurring as part of the insert operation. (Strictly speaking, it is only necessary to have a relation-high user, i.e., a user to whom all tuples are visible.) In the context of the example this means that a U-user who wishes to insert the tuple: (Enterprise, Exploration, Talos), must request a S-user to do the insertion. The S-user does so by invoking a trusted subject which can

check for key conflict, and if there is none, insert a U-tuple by writing down. If there is a conflict the S-user informs the U-user about it, so the U-user can, for example, change the name of the Starship to Enterprise'.

The first approach is available in any MLS DBMS which allows a range of access classes for individual attributes (or attribute groups), by simply limiting the classification range of the apparent key to be a singleton set. The second approach is available to any DBMS that can enforce domain constraints with adequate generality. The third approach is always available but requires the use of trusted code. Note that although there is some leakage of information, it is with a human in the loop. This type of information flow cannot be completely eliminated [1]. The best approach will depend upon the characteristics of the MLS DBMS and the application, particularly concerning the frequency and source of insertions.

The prevention approach also proposes techniques to prevent attribute polyinstantiation without compromising on confidentiality, integrity, or denial-of-service requirements. The basic idea is to introduce a special symbol denoted by "restricted" as the possible value of a data element . The value "restricted" is distinct from any other value for that element and is also different from "null." In other words the domain of a data element is its natural domain extended with "restricted" and "null." [31] then defines the semantics of "restricted" so as to be able to eliminate both visible as well as invisible polyinstantiation.

Consider again the visible polyinstantiation scenario of Chapter 2.3. Begin with the following relation:

Starship	Objective	Destination	TC
Enterprise U	Exploration U	Talos U	U

Next suppose an S-user attempts to modify the destination of the Enterprise to be Rigel. This update does not cause any security violation. But now suppose that the new destination is classified Secret. The prevention approach requires the S-user first to login as a U-user* and to mark the destination of the Enterprise as "restricted" giving the following relation:

*Alternately the S-user logs in at the U-level and requests some properly authorized U-user to carry out this step. Communication of this request from the S-user to U-user may also occur outside the computer system, say by direct personal communication or a secure telephone call

Starship	Objective	Destination	TC
Enterprise U	Exploration U	restricted U	U

The meaning of <restricted,U> is that this field can no longer be updated by an ordinary U- user.† U-users can therefore infer that the true value of Enterprise's destination is classified at some level not dominated by U. The S-user then logs in as a S-subject and enters the destination of the Enterprise as Rigel giving us the following relations at the U- and S-levels respectively:

Starship	Objective	Destination	TC
Enterprise U	Exploration U	restricted U	U

Starship	Objective	Destination	TC
Enterprise U	Exploration U	restricted U	U
Enterprise U	Exploration U	Rigel S	S

Note that this protocol does not introduce a signaling channel from an S-subject to an U- subject. There is an information flow, but from an S-user (logged in as an U-subject) to an U-subject. This is an important distinction. As mentioned in the Orange Book [1], there is the possibility that subjects may themselves constitute Trojan Horses. This type of information flow, which includes humans in the process, cannot be completely eliminated.

Next consider how the invisible polyinstantiation scenario of chapter 2.4 works with the restricted requirement. In this case the Enterprise can have a secret destination only if the destination has been marked as being restricted at the unclassified level. Thus one possibility is that the S- and U- users respectively see the following instances of SOD:

Starship	Objective	Destination	TC
Enterprise U	Exploration U	restricted U	U
Enterprise U	Exploration U	Rigel S	S

†Only those U-users with the "unrestrict" privilege for this field can update it. See [31] for details.

Starship	Objective	Destination	TC
Enterprise U	Exploration U	restricted U	U

Alternatively, both S- and U-users may see the following instance:

Starship	Objective	Destination	TC
Enterprise U	Exploration U	null U	U

In the former event an attempt by a U-user to update the destination of the Enterprise to Talos will be rejected, whereas in the latter event the update will be allowed (without causing polyinstantiation).

The concept of the "restricted" mark is straightforward, so long as the classification lattice is totally ordered. In the general case of a partially ordered lattice some subtleties arise. How to completely eliminate polyinstantiation using restricted is discussed at length in [31]. In general, updating the value of an attribute to "restricted" cannot cause polyinstantiation. On the other hand, updating the value of an attribute to a data value, say, at the C-level can be the cause of polyinstantiation. If polyinstantiation is to be completely prohibited, this update must require that the data element is restricted at all levels which do not dominate C. The fact that the data element is restricted at all levels below C can be verified by the usual integrity checking mechanisms in a DBMS [31]. However, it is tricky to guarantee this at levels incomparable with C. In preparing to enter a data value at the C level, the system would need to start a system-low (really data-element-low) process which can then write up. A protocol for this purpose is described in [31].[‡]

5.3.4 Explicit Alternatives Approach

The fourth approach described here allows the application developer to choose among explicit alternatives for polyinstantiation. In [33] Sandhu and Jajodia brought together a number of their previously published ideas, along with some new ones, to define a particular semantics for polyinstantiation called *polyinstantiation for cover stories* (PCS). PCS allows two alternatives for each attribute (or attribute group) of a multilevel tuple:

[‡]It should be noted this protocol works for arbitrary lattice, and does not require any trusted subjects. The use of trusted subjects will allow simpler protocols for this purpose.

1. No polyinstantiation, or
2. Polyinstantiation at the explicit request of a user to whom the polyinstantiation is visible.

PCS strictly limits the extent of polyinstantiation by requiring that each real-world entity be modeled in a multilevel relation by at most one tuple per security class. The goal of PCS is to provide a natural, intuitive and useful technique for implementing cover stories, with run-time flexibility regarding the use of cover stories. A particular attribute may be used for cover stories for some tuples and not for others. Even for the same real-world entity, a particular attribute may be polyinstantiated at some time and not at other times.

PCS combines the "one tuple per tuple class" concept with the "restricted" concept of Chapter 5.3.3. The basic motivation for PCS can be appreciated by considering the following instance of SOD:

Starship		Objective		Destination		TC
Enterprise	U	restricted	U	Talos	U	U
Enterprise	U	Spying	S	Rigel	S	S

In this case the Destination attribute of the Enterprise is polyinstantiated, so that <Talos,U> is a cover story for the real S destination of Rigel. The Objective is not polyinstantiated.

Consider the occurrence of polyinstantiation due to invisible polyinstantiation, as discussed by example in Chapter 2.4. This example begins with S- and U-users respectively having the following views of SOD:

Starship		Objective		Destination		TC
Enterprise	*U*	*Exploration*	*U*	*Rigel*	*S*	S

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	U	U

So far there is no polyinstantiation. Polyinstantiation occurs in the example when a U-user updates the destination of the Enterprise to be Talos.

The PCS takes a slightly different approach to this example. According to the PCS approach, polyinstantiation does exist in the S-instance of SOD given above. PCS shows this instance as:

Starship	Objective	Destination	TC
Enterprise U	Exploration U	null U	U
Enterprise U	Exploration U	Rigel S	S

In this approach, polyinstantiation already exists prior to the U-user updating the destination of the Enterprise to be Talos. This update merely modifies an already polyinstantiated relation instance to be:

Starship	Objective	Destination	TC
Enterprise U	Exploration U	Talos U	U
Enterprise U	Exploration U	Rigel S	S

With this approach, *element polyinstantiation can occur only due to visible polyinstantiation.* Invisible polyinstantiation simply cannot be the cause of element polyinstantiation. Consequently, polyinstantiation will occur only by the deliberate action of a user to whom the polyinstantiation is immediately available. In other words, polyinstantiation does not occur as a surprise.

The PCS approach treats null values like any other data value (except in the apparent key fields where "null" should not occur). Previous work on the semantics of null in polyinstantiated databases has taken the view that nulls are subsumed by non-null values independent of the access class [18, 30]. In this case the first tuple in the following relation available to S-users:

Starship	Objective	Destination	TC
Enterprise U	Exploration U	null U	U
Enterprise U	Exploration U	Rigel S	S

is subsumed by the second tuple, resulting in the following relation for S-users used in the invisible polyinstantiation example of Chapter 2.4:

Starship	Objective	Destination	TC
Enterprise U	Exploration U	Rigel S	S

Under the explicit alternative approach, the former relation is completely acceptable. The latter can be acceptable, but only if the lower limit on the classification of the destination attribute is S.

To further illustrate the semantics of null in PCS, consider the following relation:

Starship		Objective		Destination		TC
Enterprise	U	Exploration	U	null	U	U
Enterprise	U	Exploration	U	null	S	S

PCS considers this to be a polyinstantiated relation. The fact that there are nulls rather than data values in the polyinstantiated field has no bearing on the treatment of this relation. The semantics of null in [18, 30] require all null values to be classified at the level of the apparent key (U in this case), thereby deeming the second tuple illegal.

The PCS approach leaves many of the choices of whether or not to polyinstantiate to the application designer. It differentiates between updates that cannot cause polyinstantiation and those that can. The PCS design uses two different keywords (UPDATE and PUPDATE) to make the distinction explicit. The PCS approach also relies on the distinguished data value "restricted." The meaning of this data value is that users at the associated classification level may not modify the value of the restricted attribute. As in the prevention approach (chapter 5.3.3), PCS includes special privileges for imposing and lifting such restrictions.

Chapter 6

Architectural Consideration

The architecture of an MLS DBMS affects the choices of polyinstantiation strategies that are available to the database administrator (DBA). There are two fundamentally different architectural alternatives available in building an MLS DBMS. The details of these architectures are beyond the scope of this report (see [2] for more details), but we present them briefly in order to point out their implications for polyinstantiation.

Figures 6.1 and 6.2 illustrate the two approaches. Figure 6.1 illustrates the Trusted Computing Base (TCB) Subset architecture. In this architecture, data at each classification level are stored in a separate database. Users at each level interact with a separate DBMS, and each DBMS has access to all databases at its level or lower.

Figure 6.2 illustrates the Trusted Subject architecture. In this architecture, data at multiple levels is stored within the same database. Users at multiple levels interact with the same DBMS, and the DBMS is trusted to protect the data according to their classification levels.

The potential for polyinstantiation is inherent in the TCB Subset architecture. The DBMS running at the lower level has no knowledge of data stored in higher level fragments, unless all keys are classified at the same (low) level. Unless specific measures are taken to cope with the problem (as, for example in the approach described in 5.3.2), polyinstantiation due to low users cannot be prevented. Attribute polyinstantiation may be allowed by defining logical relations which span multiple levels. The underlying databases would store single- level fragments of the relations. Restrictions on fragmentation are the first method to control the types of polyinstantiation semantics allowed within a system.

Various polyinstantiation strategies have been proposed to control the recomposition of relations at the time of data retrieval. The DBMS must determine how

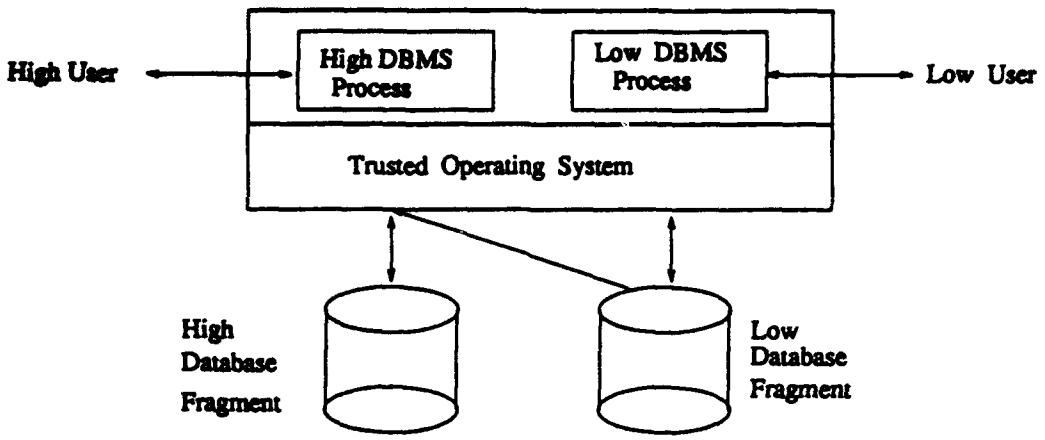

Figure 6.1: Trusted Computing Base Subset Architecture

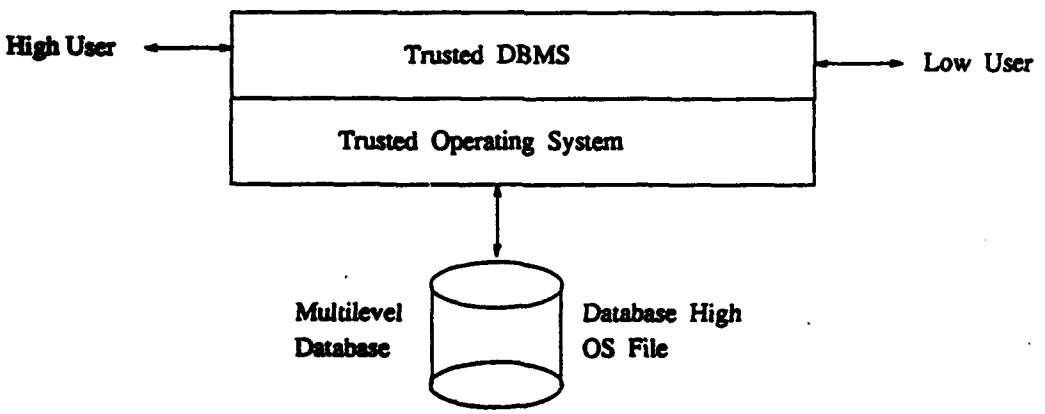

Figure 6.2: Trusted Subject Architecture

to combine the data received from the underlying databases into a single answer for the user. The approach may be to perform joins and return combinations of data (as in the SeaView approach, chapter 5.1), to choose the data with the highest classification level whenever there are polyinstantiated data (as in the MLS GDSS approach, chapter 5.2), to return data at the classification levels explicitly requested by the user (as in the belief approach, chapter 5.3.1), or some other strategy.

Under the Trusted Subject architecture, a DBA has more flexibility to trade strict security enforcement for data integrity. If the DBA chooses to use polyinstantiation rather than to permit disclosure channels, then the trusted DBMS must enforce its own barriers between data at different levels. In effect, the barriers that were imposed by the TCB Subset architecture are being reinstated through software in the trusted DBMS. Under the Trusted Subject architecture, the DBA may also choose to allow lower-level users to see some information about the existence of higher-level data in order to enforce data integrity. Since the trusted DBMS has access to data at all levels, it is able to impose restrictions on lower- level updates.

Chapter 7

Conclusion

The design of an MLS DBMS must take into account the problem of polyinstantiation. When data items exist at multiple classification levels, there is the potential for inconsistent values for the same data item at different levels. Polyinstantiation may occur over tuples or attributes, and it may arise through updates at low or high classification levels. Researchers have developed a number of different approaches to polyinstantiation; no one solution is best for all applications. This report outlined approaches in which the system

- Propagates polyinstantiated tuples to reflect valid combinations of values (chapter 5.1),
- Shows users derived tuples based on underlying polyinstantiated tuples (chapter 5.2), and
- Informs users explicitly of restrictions or inconsistencies present in the data so that polyinstantiation can be controlled (chapter 5.3).

References

[1] "Department of Defense Trusted Computer System Evaluation Criteria." Department of Defense, National Computer Security Center, (December 1985).

[2] "Multilevel Data Management Security," Committee on Multilevel Data Management Security, Air Force Studies Board, National Research Council, Washington, DC (1983).

[3] Rae K. Burns, "Referential Secrecy." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1990, pages 133-142.

[4] David D. Clark and David R. Wilson, "A Comparison of Commercial and Military Computer Security Policies." *Proc. IEEE Symposium on Security and Privacy*, April 1987, pages 184-194.

[5] C. J. Date, *An Introduction to Database Systems*. Volume II, Addison-Wesley, (1983).

[6] Dorothy E. Denning, *Cryptography and Data Security*. Addison-Wesley, Reading, Mass., (1982).

[7] Dorothy E. Denning, Teresa F. Lunt, Roger R. Schell, Mark Heckman, and William R. Shockley, "A multilevel relational data model." *Proc. IEEE Symposium on Security and Privacy*, April 1987, pages 220-234.

[8] Dorothy E. Denning, Teresa F. Lunt, Roger R. Schell, William R. Shockley, and Mark Heckman, "The SeaView security model." *Proc. IEEE Symposium on Security and Privacy*, April 1988, pages 218-233.

[9] Dorothy E. Denning, "Lessons Learned from Modeling a Secure Multilevel Relational Database System." In *Database Security: Status and Prospects*, (C. E. Landwehr, editor), North-Holland, 1988, pages 35-43.

[10] Judith N. Froscher and Catherine Meadows, "Achieving a trusted database management system using parallelism." *Database Security II: Status and Prospects*, (C. E. Landwehr, ed.), North-Holland, 1989, pages 151-160.

[11] C. Garvey, "Multilevel Data Storage Design." TRW Defense Systems Group (1986).

[12] Cristi Garvey, Thomas Hinke, Nancy Jensen, Jane Solomon, and Amy Wu, "A layered TCB implementation versus the Hinke-Schaefer approach." In *Database Security III: Status and Prospects,* (D. L. Spooner and C. E. Landwehr, eds.), North-Holland, 1990, pages 151-165.

[13] Richard Graubart, " A comparison of three secure DBMS architectures." In *Database Security III: Status and Prospects,* (D. L. Spooner and C. E. Landwehr, eds.), North-Holland, 1990, pages 109-114.

[14] Patricia P. Griffiths and Bradford W. Wade, "An authorization mechanism for a relational database system." *ACM Trans. on Database Systems*, Vol. 1, No. 3, September 1976, pages 242-255.

[15] M. J. Grohn, "A Model of a Protected Data Management System." Technical Report ESD-TR-76-289, I.P. Sharp Associates Ltd., (1976).

[16] J. T. Haigh, R. C. O'Brien, and D. J. Thomsen, "The LDV Secure Relational DBMS Model." *Database Security IV: Status and Prospects*, S. Jajodia and C. E. Landwehr (editors), North-Holland, 1991, pages 265-279.

[17] Thomas H. Hinke and Marvin Schaefer, "Secure Data Management System." Technical Report RADC-TR-75-266, System Development Corporation (1975).

[18] Sushil Jajodia and Ravi Sandhu, "Polyinstantiation integrity in multilevel relations." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1990, pages 104–115.

[19] Sushil Jajodia and Ravi Sandhu, "A formal framework for single level decomposition of multilevel relations." *Proc. IEEE Workshop on Computer Security Foundations*, Franconia, New Hampshire, June 1990, pages 152–158.

[20] Sushil Jajodia and Ravi Sandhu, "Polyinstantiation integrity in multilevel relations revisited." *Database Security IV: Status and Prospects*, S. Jajodia and C. E. Landwehr (editors), North-Holland, 1991, pages 297–307.

[21] Sushil Jajodia and Ravi Sandhu, "Database security: Current status and key issues," *ACM SIGMOD Record,* Vol. 19, No. 4, December 1990, pages 123–126.

[22] Sushil Jajodia and Ravi Sandhu, "A novel decomposition of multilevel relations into single-level relations." *Proc. IEEE Symposium on Security and Privacy*, Oakland, California, May 1991, pages 300–313.

[23] Sushil Jajodia and Ravi Sandhu, "Toward a multilevel secure relational data model," *Proc. ACM SIGMOD Int'l. Conf. on Management of Data,* Denver, Colorado, May 29-31, 1991, pp. 50–59.

[24] Sushil Jajodia and Ravi S. Sandhu, "Enforcing primary key requirements in multilevel relations," *Proc. 4th RADC Workshop on Multilevel Database Security,* Little Compton, Rhode Island, April 1991. To be published as a MITRE Techical Report.

[25] Sushil Jajodia, Ravi Sandhu, and Edgar Sibley, "Update semantics of multilevel relations." *Proc. 6th Annual Computer Security Applications Conf.*, December 1990, pages 103–112.

[26] Lunt, T.F. et al. *Secure Distributed Data Views.* Volume 1-4, SRI Project 1143, SRI International (1988-89).

[27] Teresa F. Lunt, Dorothy E. Denning, Roger R. Schell, Mark Heckman, and William R. Shockley, "The SeaView security model." *IEEE Transactions on Software Engineering,* Vol. 16, No. 6, June 1990, pages 593-607.

[28] Teresa F. Lunt and Donovan Hsieh, "Update semantics for a multilevel relational database." *Database Security IV: Status and Prospects,* S. Jajodia and C. E. Landwehr, (editors), North-Holland, 1991, pages 281-296.

[29] Doug Nelson and Chip Paradise, "Using polyinstantiation to develop an MLS application," *Proc. IEEE 7th Annual Computer Security Applications Conference,* December 1991, pages 12–22.

[30] Ravi Sandhu, Sushil Jajodia, and Teresa Lunt, "A new polyinstantiation integrity constraint for multilevel relations." *Proc. IEEE Workshop on Computer Security Foundations,* Franconia, New Hampshire, June 1990, pages 159–165.

[31] Ravi Sandhu and Sushil Jajodia, "Honest databases that can keep secrets," *Proc. 14th NIST-NCSC National Computer Security Conference,* Washington, D.C., October 1991, pages 267–282.

[32] Ravi S. Sandhu and Sushil Jajodia, "Eliminating polyinstantiation securely," *Computers & Security,* Vol. 11, 1992, pages 547-562.

[33] Ravi S. Sandhu and Sushil Jajodia, "Polyinstantiation for cover stories," *Proc. European Symp. on Research in Computer Security,* Toulouse, France, Springer-Verlag Lecture Notes in Computer Science, Vol. 648, 1992, pages 307–328.

[34] M. Schkolnick and P. Sorenson, "The Effects of Denormalization on Database Performance." *The Australian Computer Journal*, Vol. 14, No. 1, February 1982, pages 12-18.

[35] Ken Smith and Marianne Winslett, "Entity modeling in the MLS relational model," *Proc. 18th Int'l. Conf. on Very Large Data Bases,* August 1992, pages 199-210.

[36] Paul D. Stachour and Bhavani Thuraisingham, "Design of LDV: A multilevel secure relational database management system." *IEEE Trans. on Knowledge and Data Engineering*, Vol. 2, No. 2, June 1990, pages 190-209.

[37] D. C. Tsichritzis and F. H. Lochovsky, *Data Models.* Prentice-Hall, (1982).

[38] S. R. Wiseman, "On the Problem of Security in Data Bases." In *Database Security III: Status and Prospects*, (Spooner, D.L. and Landwehr, C.E., editors), North-Holland, 1990 pages 143-150.

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence; Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.